Mensagens HTTP

As mensagens HTTP são o mecanismo utilizado para trocar dados entre um servidor e um cliente no protocolo HTTP. Existem dois tipos de mensagens: **requisições**, enviadas pelo cliente para acionar uma ação no servidor, e **respostas**, a resposta que o servidor envia em resposta a uma requisição.

Desenvolvedores raramente, se é que alguma vez, constroem mensagens HTTP do zero. Aplicações como um navegador, proxy ou servidor web usam softwares projetados para criar mensagens HTTP de maneira confiável e eficiente. A forma como as mensagens são criadas ou transformadas é controlada por meio de APIs em navegadores, arquivos de configuração para proxies ou servidores, ou outras interfaces.

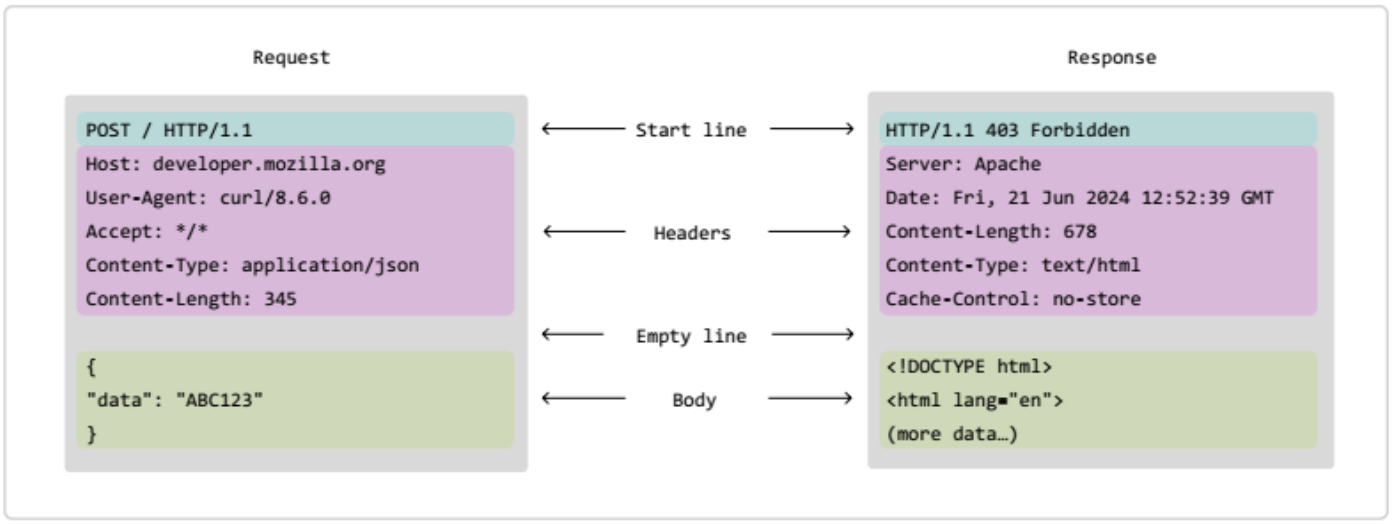
Nas versões do protocolo HTTP até o HTTP/2, as mensagens são baseadas em texto e são relativamente simples de ler e entender depois que você se familiariza com o formato. No HTTP/2, as mensagens são encapsuladas em molduras binárias (*binary framing*), o que as torna um pouco mais difíceis de ler sem certas ferramentas. No entanto, a semântica subjacente do protocolo é a mesma, então você pode aprender a estrutura e o significado das mensagens HTTP com base no formato textual das mensagens HTTP/1.x e aplicar esse entendimento ao HTTP/2 e além.

Este guia usa mensagens HTTP/1.1 para facilitar a leitura e explica a estrutura das mensagens HTTP usando o formato HTTP/1.1. Destacamos algumas diferenças que você pode precisar para descrever o HTTP/2 na seção final.

Anatomia de uma mensagem HTTP

Nota: Você pode ver mensagens HTTP na aba de *Network* das ferramentas de desenvolvedor de um navegador ou se imprimir mensagens HTTP no console usando ferramentas de linha de comando como o curl, por exemplo.

Para entender como as mensagens HTTP funcionam, vamos observar mensagens HTTP/1.1 e examinar a estrutura. A ilustração a seguir mostra como as mensagens no HTTP/1.1 se parecem:



Tanto requisições quanto respostas compartilham uma estrutura semelhante:

1. Uma **linha inicial** é uma única linha que descreve a versão HTTP junto com o método da requisição ou o resultado da requisição.
2. Um conjunto opcional de **cabeçalhos HTTP** contendo metadados que descrevem a mensagem. Por exemplo, uma requisição por um recurso pode incluir os formatos permitidos desse recurso, enquanto a resposta pode incluir cabeçalhos para indicar o formato real retornado.
3. Uma **linha em branco**, indicando que os metadados da mensagem estão completos.
4. Um **corpo opcional** contendo dados associados à mensagem. Isso pode ser dados de um POST a serem enviados ao servidor em uma requisição, ou algum recurso retornado ao cliente em uma resposta. Se uma mensagem contém ou não um corpo é determinado pela linha inicial e pelos cabeçalhos HTTP.

A linha inicial e os cabeçalhos da mensagem HTTP são coletivamente conhecidos como o **cabeçalho da requisição**, e a parte posterior que contém o conteúdo é conhecida como o **corpo**.

Requisições HTTP

Vamos observar o seguinte exemplo de requisição HTTP **POST** que é enviada após um usuário enviar um formulário em uma página da web:

```makefile
POST /users HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 49

name=FirstName+LastName&email=bsmth%40example.com
```

A **linha inicial** em requisições HTTP/1.x (POST /users HTTP/1.1 no exemplo acima) é chamada de **linha de requisição** (*request-line*) e é composta de três partes:

- O **método HTTP** (também conhecido como verbo HTTP) é um conjunto definido de palavras que descreve o significado da requisição e o resultado desejado. Por exemplo, GET indica que o cliente gostaria de receber um recurso em retorno, e POST significa que o cliente está enviando dados para um servidor.
- O **alvo da requisição** (*request target*) é geralmente uma URL absoluta ou relativa, e é caracterizado pelo contexto da requisição. O formato do alvo da requisição depende do método HTTP usado e do contexto da requisição. Isso é descrito em mais detalhes na seção **Alvos de requisição** abaixo.
- A **versão HTTP**, que define a estrutura da mensagem restante, atuando como um indicador da versão esperada a ser usada na resposta. Isso é quase sempre HTTP/1.1, já que HTTP/0.9 e HTTP/1.0 são obsoletos. Em HTTP/2 e versões superiores, a versão do protocolo não está incluída nas mensagens, pois ela é compreendida a partir do estabelecimento da conexão.

Alvos de requisição

Existem algumas formas de descrever um alvo de requisição, mas de longe a mais comum é a **forma origin**. Aqui está uma lista dos tipos de alvos e quando eles são usados:

- Na **forma origin**, o destinatário combina um caminho absoluto com as informações no cabeçalho Host. Uma string de consulta (*query string*) pode ser adicionada ao caminho para informações adicionais (geralmente no formato chave=valor). Isso é usado com os métodos GET, POST, HEAD e OPTIONS:

```swift
GET /en-US/docs/Web/HTTP/Guides/Messages HTTP/1.1
```

- A **forma absoluta** é uma URL completa, incluindo a autoridade, e é usada com GET ao se conectar a um proxy:

```makefile
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
```

```bash
GET https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Messages HTTP/1.1
```

- A **forma authority** é a autoridade e a porta separadas por dois-pontos (:). É usada apenas com o método CONNECT ao configurar um túnel HTTP:

```sql
CONNECT developer.mozilla.org:443 HTTP/1.1
```

- A **forma asterisco** é usada somente com OPTIONS quando você deseja representar o servidor como um todo (*) em vez de um recurso nomeado:

```nginx
OPTIONS * HTTP/1.1
```

Cabeçalhos de requisição

Os **cabeçalhos** são metadados enviados com uma requisição após a linha inicial e antes do corpo.
No exemplo de envio de formulário acima, eles são as seguintes linhas da mensagem:

```makefile
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
```

Em HTTP/1.x, cada cabeçalho é uma string insensível a maiúsculas e minúsculas seguida por dois-pontos (:) e um valor cujo formato depende do cabeçalho. O cabeçalho inteiro, incluindo o valor, consiste em uma única linha. Essa linha pode ser bastante longa em alguns casos, como o cabeçalho Cookie.

Alguns cabeçalhos são usados exclusivamente em requisições, enquanto outros podem ser enviados em requisições e respostas, ou podem ter uma categorização mais específica:

- **Cabeçalhos de requisição** fornecem contexto adicional para uma requisição ou adicionam lógica extra sobre como ela deve ser tratada por um servidor (por exemplo, requisições condicionais).
- **Cabeçalhos de representação** são enviados em uma requisição se a mensagem tiver um corpo, e descrevem a forma original dos dados da mensagem e qualquer codificação aplicada. Isso permite que o destinatário entenda como reconstruir o recurso como ele era antes de ser transmitido pela rede.

Corpo da requisição

O **corpo da requisição** é a parte de uma requisição que carrega informações para o servidor.
Apenas as requisições PATCH, POST e PUT têm corpo.
No exemplo de envio de formulário, esta parte é o corpo:

```ini
name=FirstName+LastName&email=bsmth%40example.com
```

O corpo da requisição no exemplo contém uma quantidade relativamente pequena de informação como pares chave=valor, mas um corpo de requisição pode conter outros tipos de dados que o servidor espera:

```perl
{
  "firstName": "Brian",
  "lastName": "Smith",
  "email": "bsmth@example.com",
  "more": "data"
}
```

Ou dados em várias partes:

```pgsql
--delimiter123
Content-Disposition: form-data; name="field1"

value1
--delimiter123
Content-Disposition: form-data; name="field2"; filename="example.txt"

Text file contents
--delimiter123--
```

Respostas HTTP

Respostas são as mensagens HTTP que um servidor envia de volta como resposta a uma requisição. A resposta permite que o cliente saiba qual foi o resultado da requisição.
Aqui está um exemplo de resposta HTTP/1.1 a uma requisição **POST** que criou um novo usuário:

```pgsql
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://example.com/users/123

{
  "message": "New user created",
  "user": {
    "id": 123,
    "firstName": "Example",
    "lastName": "Person",
    "email": "bsmth@example.com"
  }
}
```

A **linha inicial** (HTTP/1.1 201 Created acima) é chamada de **linha de status** (*status line*) em respostas e possui três partes:

- A **versão HTTP** da mensagem restante.
- Um **código de status** numérico que indica se a requisição foi bem-sucedida ou falhou.
  Códigos comuns são 200, 404 ou 302.
- O **texto de status** é uma breve descrição textual, puramente informacional, do código de status para ajudar um humano a entender a mensagem HTTP.

Cabeçalhos de resposta

Cabeçalhos de resposta são os metadados enviados com uma resposta. Em HTTP/1.x, cada cabeçalho é uma string insensível a maiúsculas e minúsculas, seguida por dois-pontos (:) e um valor cujo formato depende de qual cabeçalho está sendo usado.

Assim como os cabeçalhos de requisição, há muitos diferentes cabeçalhos que podem aparecer em respostas, e eles são categorizados como:

- **Cabeçalhos de resposta**, que fornecem contexto adicional sobre a mensagem ou adicionam lógica extra sobre como o cliente deve fazer requisições subsequentes.
  Por exemplo, cabeçalhos como Server incluem informações sobre o software do servidor, enquanto Date indica quando a resposta foi gerada.
  Há também informações sobre o recurso sendo retornado, como seu tipo de conteúdo (Content-Type) ou como ele deve ser armazenado em cache (Cache-Control).

- **Cabeçalhos de representação** — se a mensagem tiver um corpo, eles descrevem a forma dos dados da mensagem e qualquer codificação aplicada.
  Por exemplo, o mesmo recurso pode ser formatado em um tipo de mídia específico como XML ou JSON, localizado para um idioma escrito ou região geográfica, e/ou comprimido ou de outra forma codificado para transmissão. Isso permite que um destinatário entenda como reconstruir o recurso como ele era antes de ser transmitido pela rede.

Corpo da resposta

Um **corpo de resposta** está incluído na maioria das mensagens ao responder a um cliente.
Em requisições bem-sucedidas, o corpo da resposta contém os dados que o cliente solicitou em uma requisição GET.
Se houver problemas com a requisição do cliente, é comum que o corpo da resposta descreva por que a requisição falhou e dê uma dica sobre se é uma falha permanente ou temporária.

Os corpos de resposta podem ser:

- **Corpos de recurso único**, definidos pelos dois cabeçalhos: Content-Type e Content-Length, ou de comprimento desconhecido e codificados em partes com Transfer-Encoding definido como chunked.

- **Corpos de múltiplos recursos**, consistindo em um corpo que contém múltiplas partes, cada uma contendo uma peça diferente de informação. Corpos *multipart* estão tipicamente associados a formulários HTML, mas também podem ser enviados em resposta a requisições Range.

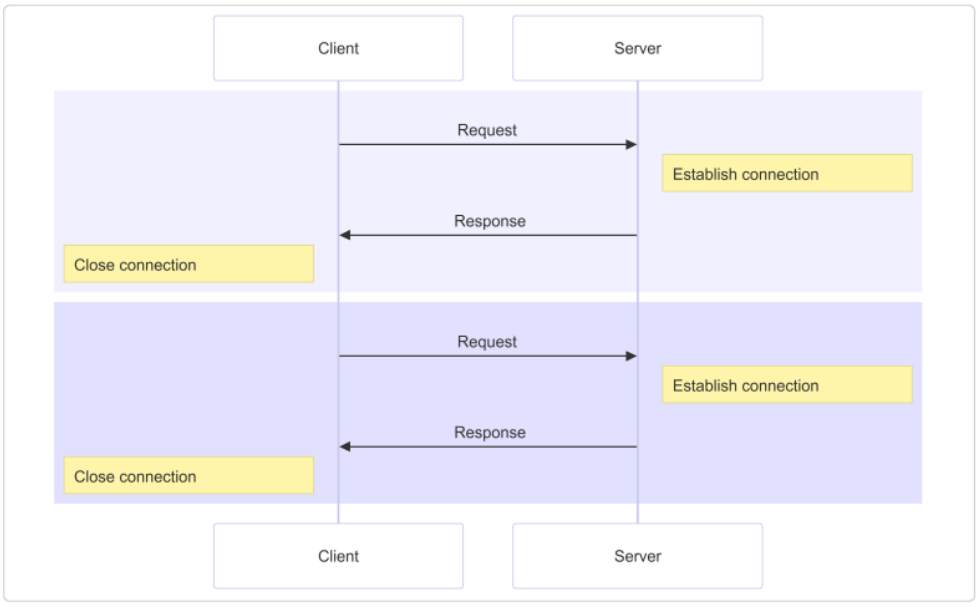
Respostas com um código de status que responde à requisição sem a necessidade de incluir conteúdo de mensagem — como 201 Created ou 204 No Content — **não possuem corpo**.

Mensagens HTTP/2

O **HTTP/1.x** usa mensagens baseadas em texto que são diretas de ler e construir, mas como resultado têm algumas desvantagens.
Você pode comprimir os corpos das mensagens usando gzip ou outros algoritmos de compressão, mas **não os cabeçalhos**.
Os cabeçalhos costumam ser semelhantes ou idênticos em uma interação cliente-servidor, mas são repetidos em mensagens sucessivas em uma conexão. Existem muitos métodos conhecidos para comprimir texto repetitivo que são muito eficientes, o que deixa uma grande quantidade de economia de largura de banda não aproveitada.

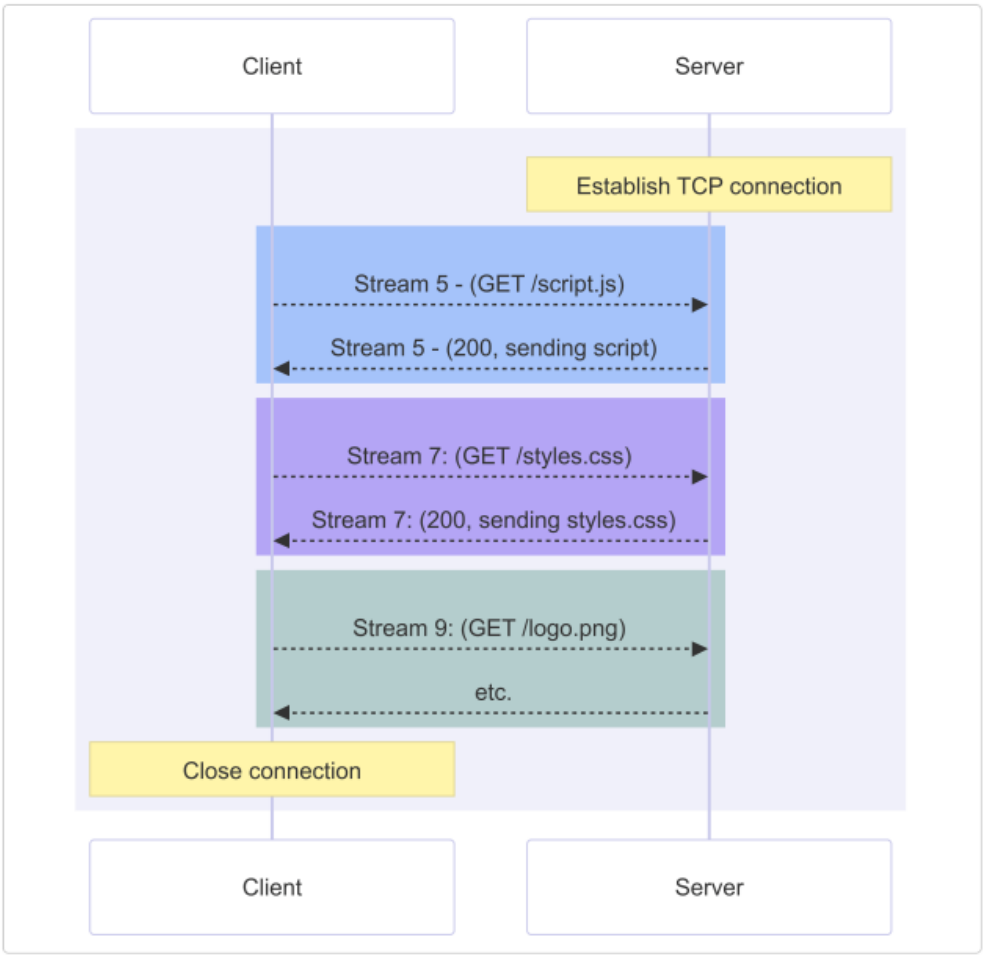
O HTTP/1.x também tem um problema chamado **bloqueio de início de linha** (*head-of-line blocking*), onde um cliente precisa esperar por uma resposta do servidor antes de enviar a próxima requisição. O *pipelining* HTTP tentou contornar isso, mas o suporte fraco e a complexidade significam que ele raramente é usado e é difícil de ser implementado corretamente.
Várias conexões precisam ser abertas para enviar requisições simultaneamente; e conexões "aquecidas" (estabelecidas e ocupadas) são mais eficientes do que conexões "frias", devido ao início lento do TCP (*TCP slow start*).

Em HTTP/1.1, se você quiser fazer duas requisições em paralelo, você precisa abrir duas conexões:



Isso significa que navegadores são limitados no número de recursos que podem baixar e renderizar ao mesmo tempo, o que normalmente foi limitado a 6 conexões paralelas.

O **HTTP/2** permite que você use uma **única conexão TCP** para múltiplas requisições e respostas ao mesmo tempo.
Isso é feito encapsulando as mensagens em uma estrutura binária e enviando as requisições e respostas em um **fluxo numerado** sobre a conexão.
Dados e cabeçalhos são tratados separadamente, o que permite que os cabeçalhos sejam comprimidos via um algoritmo chamado **HPACK**.
Usar a mesma conexão TCP para lidar com múltiplas requisições ao mesmo tempo é chamado de **multiplexação**.



As requisições não são necessariamente sequenciais: o **stream 9** não precisa esperar o **stream 7** terminar, por exemplo.

Os dados de múltiplos fluxos são geralmente entrelaçados (*interleaved*) na conexão, então os streams 9 e 7 podem ser recebidos pelo cliente ao mesmo tempo.

Existe um mecanismo no protocolo para definir uma **prioridade para cada fluxo ou recurso**.
Recursos com prioridade baixa consomem menos largura de banda do que recursos com prioridade mais alta quando estão sendo enviados por fluxos diferentes, ou podem ser efetivamente enviados sequencialmente na mesma conexão se houver recursos críticos que devem ser tratados primeiro.

Em geral, apesar de todas as melhorias e abstrações adicionadas ao HTTP/1.x, praticamente **nenhuma mudança é necessária** nas APIs usadas por desenvolvedores para utilizar o HTTP/2 sobre o HTTP/1.x.
Quando o HTTP/2 está disponível tanto no navegador quanto no servidor, ele é ativado e usado automaticamente.

Pseudo-cabeçalhos

Uma mudança notável nas mensagens no HTTP/2 é o uso de **pseudo-cabeçalhos**.
Enquanto o HTTP/1.x utilizava a **linha inicial da mensagem**, o HTTP/2 usa campos especiais de pseudo-cabeçalho que começam com :.
Nas requisições, existem os seguintes pseudo-cabeçalhos:

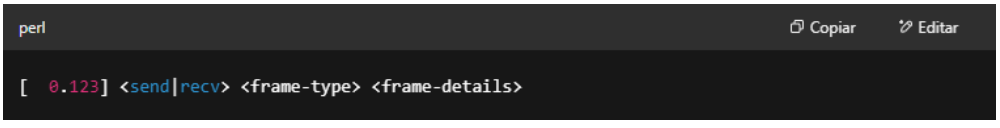
- :method – o método HTTP.
- :scheme – a parte do esquema da URI de destino, que geralmente é HTTP(S).
- :authority – a parte da autoridade da URI de destino.
- :path – o caminho e partes de consulta da URI de destino.

Nas respostas, existe apenas um pseudo-cabeçalho, que é:

- :status – fornece o código da resposta.

Podemos fazer uma requisição HTTP/2 usando o **nghttp** para buscar example.com, o que imprimirá a requisição em um formato mais legível.
Você pode fazer a requisição usando este comando, onde a opção -n descarta os dados baixados e -v significa *verbose* (detalhado), mostrando a recepção e transmissão das frames:

Se você observar na saída, verá o tempo para cada frame transmitido e recebido:



Não precisamos entrar em muitos detalhes sobre esta saída, mas procure pela frame HEADERS no formato:

Nas linhas após a transmissão do cabeçalho, você verá as seguintes linhas:

```makefile
:method: GET
:path: /
:scheme: https
:authority: www.example.com
accept: */*
accept-encoding: gzip, deflate
user-agent: nghttp2/1.61.0
```

Isso deve parecer familiar se você já estiver confortável trabalhando com HTTP/1.x, e os conceitos abordados nas seções anteriores deste guia ainda se aplicam.
Este é o frame binário com a requisição GET para example.com, convertido em uma forma legível pelo **nghttp**.

Se você observar mais abaixo na saída do comando, verá o pseudo-cabeçalho :status em um dos fluxos recebidos do servidor:

```perl
[  0.433] recv (stream_id=13) :status: 200
[  0.433] recv (stream_id=13) content-encoding: gzip
[  0.433] recv (stream_id=13) age: 112721
[  0.433] recv (stream_id=13) cache-control: max-age=604800
[  0.433] recv (stream_id=13) content-type: text/html; charset=UTF-8
[  0.433] recv (stream_id=13) date: Fri, 13 Sep 2024 12:56:07 GMT
[  0.433] recv (stream_id=13) etag: "3147526947+gzip"
```

E se você remover o tempo e o ID do stream dessa mensagem, ela deve parecer ainda mais familiar:

```makefile
:status: 200
content-encoding: gzip
age: 112721
```

Aprofundar-se em frames de mensagens, IDs de stream e como a conexão é gerenciada está além do escopo deste guia, mas para o propósito de entender e depurar mensagens HTTP/2, você deve estar bem preparado usando o conhecimento e ferramentas deste artigo.

Conclusão

Este guia fornece uma visão geral da anatomia das mensagens HTTP, usando o formato HTTP/1.1 para ilustração. Também exploramos o encapsulamento das mensagens HTTP/2, que introduz uma camada entre a sintaxe do HTTP/1.x e o protocolo de transporte subjacente, sem modificar fundamentalmente a semântica do HTTP.

O HTTP/2 foi introduzido para resolver os problemas de bloqueio de início de linha (*head-of-line blocking*) presentes no HTTP/1.x, permitindo a multiplexação das requisições.

Um problema que permaneceu no HTTP/2 é que, embora o bloqueio de início de linha tenha sido resolvido no nível do protocolo, ainda há um gargalo de desempenho devido ao bloqueio dentro do **TCP** (no nível de transporte).
O **HTTP/3** resolve essa limitação usando o **QUIC**, um protocolo construído sobre **UDP**, em vez de TCP.
Essa mudança melhora o desempenho, reduz o tempo de estabelecimento de conexão e aumenta a estabilidade em redes degradadas ou instáveis.
O HTTP/3 mantém as mesmas semânticas centrais do HTTP, portanto características como métodos de requisição, códigos de status e cabeçalhos permanecem consistentes entre as três principais versões do HTTP.

Se você entende a semântica do HTTP/1.1, já tem uma base sólida para compreender o HTTP/2 e o HTTP/3.
A principal diferença está em como essas semânticas são implementadas no nível de transporte.

Seguindo os exemplos e conceitos deste guia, você agora deve estar preparado para trabalhar com HTTP e entender o significado das mensagens e como as aplicações usam HTTP para enviar e receber dados.